2020

Stream Clustering And Visualization Of Geotagged Text Data For Crisis Management

Nathaniel C. Crossman
Wright State University

STREAM CLUSTERING AND VISUALIZATION OF GEOTAGGED TEXT DATA FOR CRISIS MANAGEMENT

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science

By

NATHANIEL C. CROSSMAN
B.S.C.S., Wright State University, 2018

2020
Wright State University

WRIGHT STATE UNIVERSITY
GRADUATE SCHOOL

04/28/2020

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY Nathaniel C. Crossman ENTITLED Stream Clustering And Visualization Of Geotagged Text Data For Crisis Management BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Master of Science.

_____
Soon M. Chung, Ph.D.
Thesis Director

_____
Mateen Rizki, Ph.D.
Chair, Department of Computer Science and Engineering

Committee on Final Examination

_____
Soon M. Chung, Ph.D.

_____
Nikolaos Bourbakis, Ph.D.

_____
Vincent A. Schmidt, Ph.D.

_____
Barry Milligan, Ph.D.
Interim Dean of the Graduate School

ABSTRACT

Crossman, Nathaniel C. Department of Computer Science and Engineering, Wright State University, 2020. Stream Clustering and Visualization of Geotagged Text Data for Crisis Management.

In the last decade, the advent of social media and microblogging services have inevitably changed our world. These services produce vast amounts of streaming data, and one of the most important ways of analyzing and discovering interesting trends in the streaming data is through clustering. In clustering streaming data, it is desirable to perform a single pass over incoming data, such that we do not need to process old data again, and the clustering model should evolve over time not to lose any important feature statistics of the data. In this research, we have developed a new clustering system that clusters social media data based on their textual content and displays the clusters and their locations on the map. Our system takes advantage of a text stream clustering algorithm, which uses the two-phase clustering process. The online micro-clustering phase incrementally creates micro-clusters, called text droplets, that represent enough information about topics occurring in the text stream. The off-line macro-clustering phase clusters micro-clusters for a user-specified time interval and can change macro-clustering algorithms dynamically. Our experiments demonstrated that the performance of our system is scalable; and it can be easily used by first responders and crisis management personnel to quickly determine if a crisis is happening, where it is concentrated, and what resources are best to deploy to the situation.

TABLE OF CONTENTS

TABLE OF FIGURES

ACKNOWLEDGEMENTS

I would like to thank my thesis advisor, Dr. Soon Chung, for his ideas, advice, guidance, and constant encouragement. Without Dr. Chung strongly suggesting I apply to DAGSI program, I would have never considered doing a thesis project or even graduated with a master's degree. Thank you!

Another important thanks are for my thesis committee members: Drs. Nikolaos Bourbakis and Vincent Schmidt for their vital time spent being committee members, and for their help in improving this thesis document.

Next, I want to thank my friend and classmate, Sri Jitendra Satpathy for always being willing to debate with me on my convoluted ideas; and helping to keep me on track, focused on the goal at hand. Additionally, a huge thanks goes out to my family and girlfriend, many of whom have helped me throughout my academic career in innumerable ways. Without their help I would never have achieved any of my goals. Thank you!

Lastly, this material is based on research sponsored by the Ohio Department of Higher Education and the Southwestern Council for Higher Education under Ohio House Bill 49 of the 132nd General Assembly. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

In short, this small accomplishment would not have been possible without numerous people helping me along my journey. Thank you, each and every one of you.

DISCLAIMER

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Southwestern Council for Higher Education, the Ohio Department of Higher Education or the U.S. Government.

CHAPTER 1
INTRODUCTION

1.1 Motivation

In the last decade, we have seen an enormous increase in disasters. Some of these disasters were natural and others were man-made moments of extreme terror. Yet crisis management personnel do not fully utilize all available resources, and they even rely on outdated risk and crisis communication models [18]. Time and time again, we see social media and microblogging services being the first to recognize that a crisis has begun and are often in the front lines communicating what is happening. Many of these microblogging services share and exchange information on a local and global scale. Often traditional news outlets rely on gathering information from these sources prior to breaking news. For example, during 2010 Haitian earthquake, microblogging services and social media helped to broadcast relevant information to first responders and other officials [3].

Subsequently, it is high time that we fully utilize the invaluable information social media and microblogging services provide us and present it to first responders and crisis management personnel in a manner that allows them to quickly decide if a crisis is occurring and how best to respond.

1.2 Previous Work

Previously, Barnard, Chung, and Schmidt [23] addressed this problem by developing a tool that automatically clusters geotagged text messages. Their goal was to create a system that allows first responders and crisis management personnel to seamlessly evaluate whether a crisis is occurring. They did this by continuously clustering text messages based on their content within a given time-frame and displaying the clusters on an interactive real-time map. They used the silhouette coefficient [27] to determine the optimal number of clusters

for each iteration of the k-means clustering algorithm.

Their system could perform well at detecting topic changes in the sequence of text messages. However, they did not consider the infinite nature of streaming data, the detection and removal of outliers, and the removal of clusters themselves when they are no longer relevant. Moreover, if a user wants to see the clustering result for a different time period, the system must process the corresponding part of the dataset all over again. This is extremely wasteful and impractical in a real-time system. In short, the only thing they really considered was the real-time clustering and visualization of messages during a relatively short time interval.

In this research, we have enhanced their system to perform text stream clustering by using the two-phase clustering approach. Moreover, we have completely reengineered the system from the bottom up to support our new system architecture.

1.3 Organization

This thesis is organized as follows:

Chapter 1 overviews the motivation and the background behind this thesis project by outlining the problem which this thesis will address. Finally this chapter discusses the previous work done to address this problem.

Chapter 2 overviews the main concepts and principles relating to this thesis, including text preprocessing, document representation, similarity measures, and internal evaluation measures. Additionally, Chapter 2 introduces stream clustering concepts and rationale behind choosing a stream clustering algorithm. Finally, this chapter concludes with a discussion and introduction to two important stream clustering algorithms.

Chapter 3 is fully devoted to presenting the text stream clustering we use for our micro-clustering system.

Chapter 4 introduces our crisis management system, our implementation, and our evaluation.

Chapter 5 introduces our current, initial visualization component and explains how it could be utilized to detect a developing crisis.

Chapter 6 is the conclusion of this master thesis and officially ends with a brief discussion on our further research plans for our crisis management system.

CHAPTER 2
BACKGROUND &
BASIC CONCEPTS ABOUT STREAM CLUSTERING

2.1 Introduction

This chapter introduces the key concepts and definitions that are crucial in order for the reader to have a clear understanding of how our crisis management system works. The techniques discussed in Chapter 2 are as follows: Text Preprocessing, Text Document Representation, Similarity Measure, and Intrinsic Evaluation Coefficients. Additionally, this chapter explains why stream clustering is important and introduces two stream clustering algorithms: Clustream and Constream. If the reader is already familiar with these concepts, they may skip to Chapter 3.

2.2 Text Preprocessing

Text preprocessing is the first crucial step in a Natural Language Processing (NLP) system, with major ramifications on the final performance of the system. Therefore, to ensure efficiency in our system, we have utilized industry-standard text processing techniques, particularly tokenizing, case-folding and stop-word removal. Following subsections are designed to give the reader a brief insight on our text preprocessing procedures.

Tokenization: is the task of chopping a character sequence (text document) into meaningful pieces, called tokens, at the same time throwing away certain characters that are irrelevant, such as punctuation. See Figure 1 for an example of tokenization.

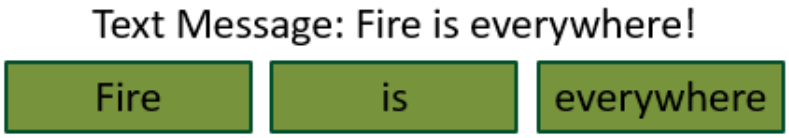

Figure 1: demonstrates a text message being broken up into individual tokens and punctuation is removed.

Case-Folding: is a common strategy when normalizing a text document by reducing all letters (terms) to lowercase. This strategy allows us to match instances of *Fire* at the beginning of a text message to instances of *fire* appearing elsewhere. This technique is extremely important in our system because most users of microblogging services often refrain from using proper capitalization.

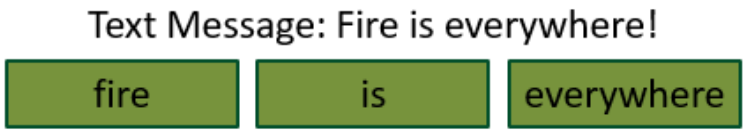

Figure 2: demonstrates individual terms the reduced to lowercase

Stop-Word Removal: is the process of dropping or removing common words. This process is extremely important for performing TF-IDF as leaving the stop-words in will not give good results. The importance of removing stop-words is better explained in section 2.3. Moreover, by removing stop-words we are drastically increasing performance.

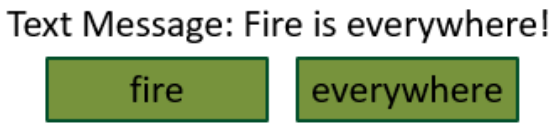

Figure 3: demonstrates stop word removal

2.3 Text Document Representation

Choosing the right document representation for text clustering is extremely important to get efficient and meaningful results. The representation we chose is called Term Frequency - Inverse Document Frequency (TF-IDF), and was originally introduced in [28]. Instead of just representing a term in a document by its raw frequency or just by its relative frequency, each term, in this representation, is weighted by dividing the term frequency (TF) by the number of documents in a corpus containing that term. This method ensures

that we avoid a common pitfall in text analysis, which happens when the most frequently used word in a document are also, often, the most frequently used words in the entire corpus. In contrast, terms with the highest TF-IDF score are the terms in a document that are distinctively frequent in a document – when that document is compared to other documents [28]. See Figure 4 for an illustration.

Documents

Term	Term #
fire	2
is	2
everywhere	1

Term	Term #
brushfire	3
is	2
everywhere	2

	Calculation for Term "is"	Calculation for Term "fire"
d1	$tf(\text{"is"}, d1) = 2$	$tf(\text{"fire"}, d1) = 2$
	$tf(\text{"is"}, d2) = 2$	$tf(\text{"fire"}, d2) = 0$
	$idf(\text{"is"}, D) = \log(\frac{2}{2}) = 0$	$idf(\text{"fire"}, D) = \log(\frac{2}{1}) = 0.301$
d2	$tf\text{-}idf(\text{"is"}, d1, D) = 2 \times 0 = 0$	$tf\text{-}idf(\text{"fire"}, d1, D) = 2 \times 0.301 = 0.602$
	$tf\text{-}idf(\text{"is"}, d2, D) = 2 \times 0 = 0$	$tf\text{-}idf(\text{"fire"}, d2, D) = 0 \times 0.301 = 0$

Figure 4: demonstrates a simplistic example of TF-IDF

In the above example, first we assign a specific weight to each term for all the documents. The weight for each term strictly depends on the number of occurrences that the term appears in the document. Simply put, we count the number of times each term appears in the document, and this weighting scheme is called term frequency and is denoted as $TF_{t,d}$.

However, this approach suffers from a critical problem: extremely common terms, such as stop-words, cause lots of documents in a corpus to appear similar to each other. Additionally, simply removing the stop-words from all documents would not entirely fix this problem. For example, if we analyze a corpus of Stack Overflow post, the word "code" would fall into the category of common terms and potentially skew our results. This is why using raw term frequency by itself is not a good solution.

To address this problem, we use the inverse document frequency (IDF), represented as follows:

$$IDF_t = \log\left(\frac{\mathrm{N}}{DF_t}\right) \quad (1)$$

where document frequency DF_t is defined as the number of documents in the corpus that contain a term *t*, N is the total number of documents in the corpus.

Now by combining the two definitions, term frequency and inverse document frequency, we produce a new and a more robust weighting scheme called TF-IDF. A typical TF-IDF mathematical representation of a document *d* is as follows:

$$d_{TF-IDF} = \left[TF_1 \log\left(\frac{\mathrm{N}}{DF_1}\right),\ TF_2 \log\left(\frac{\mathrm{N}}{DF_2}\right),\ \ldots,\ TF_w \log\left(\frac{\mathrm{N}}{DF_W}\right)\right] \quad (2)$$

where TF_i is the term frequency of a term *i* in *d*, DF_i is the number of documents containing term *i*, *W* is the total number of unique terms in the dataset, and *N* is the total number of documents in the dataset. To account for the documents of different lengths, each document vector is normalized to a unit vector (i.e., $||d_{TF-IDF}|| = 1$).

Figure 4 demonstrates that the TF-IDF score for the term "is" equals zero which indicates that this term in the collection is not informative. This is because the term appears in all the documents. On the other hand, the term "fire" has a higher TF-IDF score, indicating that it is more interesting.

2.4 Similarity Measure

Once we have calculated the TF-IDF of each document, we can calculate the similarity between the documents using the cosine similarity. The cosine similarity measures the cosine of the angle between two vectors. In our TF-IDF model, each document is represented as a discrete vector whose direction is determined by its TF-IDF values in a vector space.

The cosine similarity can be calculated for two documents represented by their TF-IDF vectors d_i and d_j as follows:

$$\cos(d_i, d_j) = \frac{d_i \bullet d_j}{\|d_i\| \|d_j\|} = \frac{\sum_{k=1}^{n} d_{i_k} d_{j_k}}{\sqrt{\sum_{k=1}^{n} d_{i_k}^2} \sqrt{\sum_{k=1}^{n} d_{j_k}^2}} \quad (3)$$

where *n* is the total number of terms, and k represents the term being iterated over. The cosine similarity is 0 for orthogonal vectors, and 1 for identical vectors. When the TF-IDF vector of each document is normalized to a vector of unit length, the above equation can be simplified to:

$$\cos(d_i, d_j) = d_i \bullet d_j = \sum_{k=1}^{n} d_{i_k} d_{j_k} \quad (4)$$

2.5 Intrinsic Evaluation Coefficients

2.5.1 Silhouette Coefficient

The silhouette coefficient is a comparative value based on the tightness and separation of the clusters created [27]. In general, it is a measure of how similar each object is to its own cluster, compared to other clusters, and it could be used to enhance the accuracy of clustering.

For each object *i*, its silhouette coefficient *s*(*i*) is defined as:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} \quad (5)$$

where *a*(*i*) is the average dissimilarity of *i* with all other objects within the same cluster, and *b*(*i*) is the lowest average dissimilarity of *i* to any other cluster that does not contain *i*. Thus, *s*(*i*) is between -1 and 1, and when it is close 1, object *i* is appropriately clustered. On the other hand, if *s*(*i*) is close to -1, object *i* better be assigned to the cluster with the lowest average dissimilarity *b*(*i*). If *s*(*i*) is close to 0, object *i* is on the border between the

two clusters [27].

2.5.2 Cophenetic Correlation Coefficient (CPCC)

CPCC can be defined as a measure of how accurately the dendrogram, resulted from aggolomerative hierarchical clustering, preserves the pairwise distance between the objects (in our case, micro-clusters). CPCC can be calculated as follows:

$$CPCC = \frac{\sum_{i=1}^{n}\sum_{j=i+1}^{n}(M_{i,j} - \bar{m})(D_{i,j} - \bar{d})}{\sqrt{\sum_{i=1}^{n}\sum_{j=i+1}^{n}(M_{i,j} - \bar{m})^2 \sum_{i=1}^{n}\sum_{j=i+1}^{n}(D_{i,j} - \bar{d})^2}} \quad (6)$$

where $M_{i,j}$ represents the distance between two objects i and j, measured in cosine similarity, and $D_{i,j}$ represents the dendrogrammatic distance between two objects i and j, which is the height of the node at which these two objects are first joined together in the dendrogram. $\bar{m}$ and $\bar{d}$ represent the average of M and D, respectively.

2.6 Why Stream Clustering?

In this section, we will briefly discuss the rationale behind choosing a stream clustering algorithm. The goal of any clustering algorithm is consistently grouping similar objects in the same cluster, whereas separating dissimilar objects to different clusters. Since the type of data can directly affect the performance and outcome of a clustering algorithm, it is imperative to find one that can efficiently work on the given data type which may have multiple dimensions.

Traditional data clustering techniques rely on the fact that the amount of data generated is not infinite, can be physically stored, and can be analyzed in incremental steps or by a batch-mode processing. They require constant re-clustering of the data and retain their results in main memory or secondary storage devices. Thus, they are extremely costly and arguably impractical for stream applications [16].

Since our goal was to build a truly practical, real-time system that could perform on a possibly unbounded text data stream of varying speed and structure, we had to choose a one-pass clustering algorithm that scans the data only once to ensure computational efficiency.

Being able to extract useful knowledge from a data stream is extremely challenging, as any stream clustering algorithm should be able to continuously cluster objects within limited memory and time constraints. Moreover, for a data stream clustering algorithm to be useful, it must consider and address the following additional constraints [8]:

1. The algorithm should have no control over the order of incoming data.
2. The algorithm should perform fast incremental processing of the data, where results are provided in a timely manner.
3. The algorithm should be able to adapt dynamically to changes in the data, which means it should be able to detect the presence of new clusters or when a cluster should be faded or removed.
4. The algorithm should be able to scale to the potentially infinite number of arriving data items.
5. The algorithm should have a model that is not only compact but does not grow with the number of individual items processed. That means, the model representation should not even consider linear growth acceptable.
6. The algorithm should have the ability to detect outliers and deal with them appropriately.

These six criteria were the cornerstone of how we ultimately decided which stream clustering algorithm to utilize for our real-time crisis management system.

2.7 Stream clustering algorithms

2.7.1 CluStream

CluStream [6] is based on the concept of micro-clusters. The micro-cluster is a data structure that captures $(2 \times dimension + 3)$ summary statistics of the data stream, including the number of data points; the sum of data points and the sum of their squares (along each dimension); and the sum of timestamps and their squares. Using the micro-clusters, we can easily perform the clustering over different time horizons (i.e., time periods).

An important property of micro-clusters is that they are additive; in other words, micro-clusters can be updated by pure additive operations. For example, if a new data point is added to a micro-cluster, each component of the micro-cluster needs to be added with the corresponding value of the data point. Similarly, the micro-clusters for a time period (t_1, t_2) can be obtained by subtracting the micro-clusters at time t_1 from those at time t_2, so that we can easily cluster the data stream during that time period (t_1, t_2). The microstructures are stored as snapshots in time that follow a pyramidal time frame [6], such that recent snapshots are stored more frequently, as shown in Figure 5.

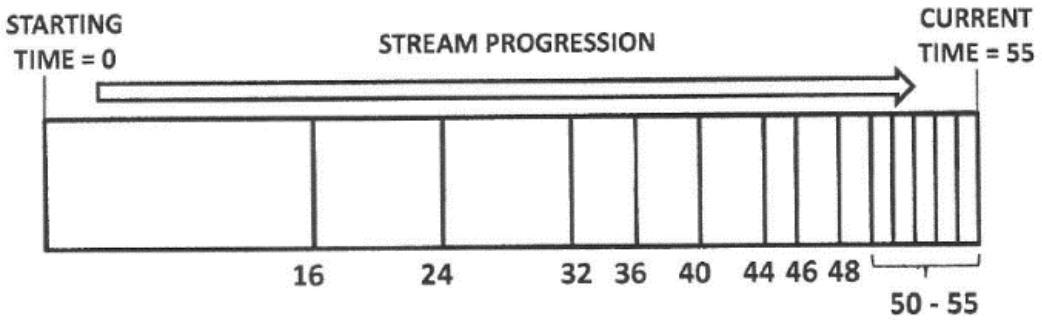

Figure 5: diagram of snapshot. (source [21])

This storage method provides an effective trade-off between the storage space requirement and the ability to recall summary statistics from different time periods, which is important

to clustering the data stream over a new time period.

However, CluStream uses the Euclidean distance for measuring similarity between data points, which does not work well for high-dimensional text documents [7]. Besides not being designed to work with textual data, CluStream has other limitations. For example, the number of micro-clusters are fixed over the course of the stream, which is not realistic in an evolving data stream. Also, CluStream produces convex-shaped clusters, as it adopts the k-means clustering paradigm, but real clusters are arbitrarily shaped. Finally, CluStream does not deal with outliers and noise, which is risky in a stream environment and might lead to the destruction of existing valid micro-clusters in order to accommodate noisy and/or outlier data points. Consequently, we decided to pursue alternative algorithms.

2.7.2 ConStream

ConStream [5] is a stream clustering algorithm that was designed to cluster text or categorical data. ConStream also uses the online and off-line approach. Moreover, the algorithm handles topic change over time and removes outliers gracefully by decaying the weights of previous clusters. In its online phase, the algorithm maintains summary statistics in the form of a cluster droplet, which can be described as a tuple containing $(\overline{DF2}, \overline{DF1}, n, w(t), l)$ at a given time *t*. For text data, $\overline{DF2}$ stores the sum of weighted counts for each pair of words in the cluster; $DF1$ stores the sum of weighted counts for each word in the cluster; n counts the number of text documents in the cluster; $w(t)$ is the weight of the cluster at time *t*; and l represents the last time the cluster was updated [5].

Unfortunately there are no available implementations of this algorithm, and we decided that the time and effort it would take to implement and test this algorithm would take away from our original goal — creating a real-time crisis management system.

Subsequently, we found an offshoot algorithm called TextClust [4] that incorporates all the benefits of ConStream and, for the most part, adheres to the above listed criteria. Moreover, TextClust has already been proved on a microblogging service and, with minor changes to their initial implementation, would work admirably in our system. A detail description of how TextClust works is given in Chapter 3.

CHAPTER 3

TextClust Algorithm

3.1 Introduction

Like most stream clustering algorithms, TextClust is built around the familiar and successful two-phase approach, composed of an online and off-line phase [4]. See Figure 6 for an illustration of the micro-clustering and macro-clustering process. Its online phase incrementally builds micro-clusters that contain just enough information to calculate a Term Frequency, Inverse Document Frequency (TF-IDF) vector for each micro-cluster. See section 2.3 for a fuller explanation of TF-IDF.

These micro-clusters represent many preliminary or potential clusters, and can also be thought of as topics, subtopics, or small discussion threads in the live stream. Each micro-cluster's TF-IDF vector is strictly maintained whenever a new text message is observed. A micro-cluster is composed of a list of *n-grams*, the occurrence counts of each *n-gram*, the micro-clusters weight, and the last time since the micro-cluster was updated [4].

Figure 6: the two-phase stream clustering process

On the other hand, the macro-clustering phase is responsible for re-clustering a group of micro-clusters that were created during a specific time period. The macro-

clustering process re-clusters these micro-clusters utilizing a traditional clustering algorithm; for example, with a hierarchical agglomerative clustering algorithm.

3.2 How Micro-Clusters Are Added

Initially the TextClust algorithm starts by tokenizing the text and building *n-grams*. Instead of using fixed size *n-gram*, users can specify a range between n_{min} and n_{max}, and the system will build all term sequences within that range. The result of this initial process is the occurrence count of each *n-gram* in the text, and it is considered as the weight of the *n-gram*. After this, a temporary micro-cluster is created and initialized for this new text. The micro-cluster is represented by the counts of *n-grams*, its weight is initialized to 1, and its time of last update is set to the current time [4].

The next step is calculating the cosine similarity between the temporary micro-cluster and each existing micro-clusters, in order to find the closest one. The counts of *n-grams* are already available, but their document frequencies should be calculated by adding their frequencies in all available micro-clusters. If the closest micro-cluster is within the user-specified distance threshold *r*, the temporary micro-cluster is merged into it; otherwise, the temporary micro-cluster is simply added as a new micro-cluster [4].

3.3 Fading Existing Micro-Clusters

No stream clustering algorithm can truly work in a real-time application if it does not, to some degree, support the fading or removal of old clusters. In this algorithm, the process of removing outdated micro-clusters is done by exponentially decaying the weights of clusters using a fading function $2^{-\lambda\Delta t}$, where λ is the user-specified fading rate and Δt is the time since the cluster was last updated.

Whenever a micro-cluster is updated (e.g., by merging), the algorithm updates its weight. Also, at every time interval t_{gap}, the algorithm executes a cleanup procedure

which decays the weight of each micro-cluster as well as the weight (i.e., frequency) of each n-gram in the micro-cluster by applying the fading function. If the weight of any micro-cluster or an n-gram falls below $2^{-\lambda t_{gap}}$, it is removed from the model. Generally, it takes a new micro-cluster at least t_{gap} time interval to decay to this weight. As a part of the cleanup procedure, every pair of micro-clusters is evaluated based on their cosine similarity to see if they should be merged together.

It should be noted, in [4], that the authors reported that fading the individual n-grams within each micro-cluster did not contribute much to the clustering quality, even though it saves the memory space.

The result of this stream clustering algorithm is a condensed list of micro-clusters that represent subtopics or discussion threads in the live text message stream.

3.4 Macro-Clustering

Upon user's requests, the macro-clustering phase performs the clustering of micro-clusters that were created during the user-specified time period. This final clustering result is often referred to as macro-clusters. This macro-clustering is also called re-clustering, and usually it is not considered as a time critical process, as it is executed as a separate process in response to a user query. This means that the macro-clustering process does not interfere with the micro-clustering process. Additionally, Macro-clustering employs a traditional clustering algorithm, as it typically processes a fixed set of micro-clusters.

CHAPTER 4

System Architecture

4.1 Introduction

In our ever-changing world, microblogging services and social media platforms endlessly broadcast data streams. These data streams exhibit temporal locality, and most batch processing clustering algorithms do not take this into account. Moreover, they don't consider the difficulty of clustering a potentially infinite stream or the detection and removal of outliers in an evolving data stream environment. It is crucial that any online clustering process not only be able to unceasingly cluster a text data stream but also provide a mechanism to analyze clusters in an off-line fashion. The reason is that users need to view data in different forms by specifying certain temporal parameter values. For example, crisis management personnel may want to view different time horizon segments of the data. Lastly, the clustering algorithm should be computationally efficient, because its processing rate should be fast enough to match the incoming rate of data [5].

To achieve this goal, we constructed a system architecture that can handle all of these and provide first responders and other users with the ability to view a developing crisis, potentially predict a crisis, view a past crisis in different time horizons, and is computationally efficient for real-time processing. Moreover, by supporting incremental clustering of incoming data, our system negates the need to process the entire dataset again in order to provide more aggregate information to the end-user. Figure 7 illustrates our system architecture.

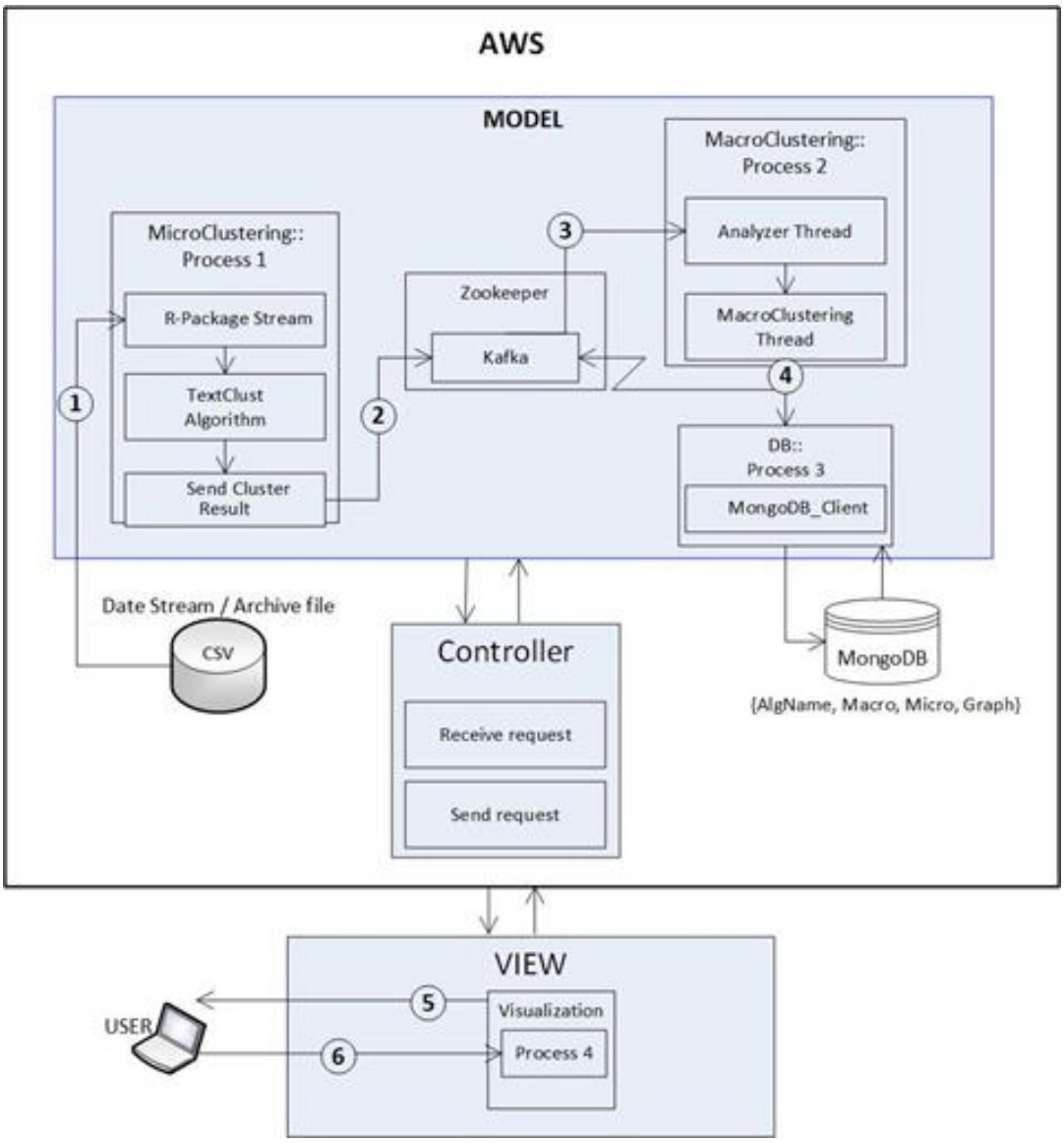

Figure 7: system architecture diagram.

4.1.1 Summary Of System Architecture

Here is a breakdown of the complete system (shown in Figure 7) and how it all works together.

1. The Stream or Archive file is incrementally read into the running **TextClust** algorithm.
2. The result of micro-clustering is sent out to the Kafka running on Zookeeper.

a. This thread also responds to the end-user requests; for example, maximum weight, n-gram size, and number of micro-clusters to retain before piping to Kafka server.

3. Kafka sends the result of micro-clustering to the analyzer thread hosted in Process 2.
    a. The analyzer thread collects the micro-clusters, and stores them “off-line” in memory.
    b. The **MacroClustering** thread accesses the shared queue containing the micro-clusters collected by the sibling analyzer thread, and uses the scikit-learn package to calculate the TF-IDF.
        i. This thread responds to user requests for changes; for example, clustering parameters and which macro-clustering algorithm to use next.
    c. The result of clustering is formatted as a JSON message.
4. The clustering thread sends the result of macro-clustering to Kafka, and also saves it on DB.
5. Kafka sends the result of macro-clustering to the node server, and the node server notifies connected client through a web socket interface that the data has been re-freshed; allowing the client to refresh the visualization.
6. Lastly, the end-user views the clustered information and interacts with the visuali-zation component.

4.2 Our System

This section will discuss our architecture and explain how our system works. A goal for the system is to make it accessible to anyone. Thus, our current system is completely open

source. The primary benefit of using open source technologies is that they are generally free, open to modification, and there is a large community of developers contributing to them.

To be a useful tool for first responders, our system needs the ability to easily upgrade over time. We also decided that our system should be built using a truly object-oriented programming language; it should promote the SOLID principles [19]; and use useful software design patterns [11]. We did all of these to ensure reuse and facilitate expansion. Additionally, to promote the reuse of our system, we followed good software engineering design principles by reducing coupling, promoting high cohesion and data-hiding. With all these criteria in mind, we chose the Python programming language.

However, this choice of programming language caused us a problem as the only implementation of the TextClust algorithm we could find was developed in the R language. Therefore, to use their implementation, we needed to call methods only available in their API and at runtime. We built a wrapper class around all-required methods; and with the help of rpy2 [9], we could invoke all of them.

> ***Note:*** *rpy2 creates a framework that allows us to translate R functions or R objects into Python functions or objects. This framework gives one the ability to call an R library function as if it is a function in Python. This is possible because rpy2 creates an embedded R process in Python.*

Moreover, we also ensured that our system architecture could easily be parallelized, conserve memory, and minimize computational cost. All the processes, shown in Figure 7, can run independently of each other and even on different servers.

To give first responders and crisis management personnel the ability to use our platform on the go, we designed our system based on the principle of a client/server

architecture and used the Model-View-Controller (MVC) design pattern [11]. We can further break this client/server architecture into three main components — view, controller, and model. The view is responsible for all visualization, receiving requests from the user, and sending them to the controller. The controller interprets all the actions that the user wants to do, and requests all the data. Finally, the model has many subcomponents that perform endless reading of the streaming data, continuous micro-clustering, user-specified macro-clustering, and saving the results for later analysis.

Thus, by developing a tool that can automatically capture data from microblogging services, authorities and dispatchers could have access to more information, almost as soon as an event happens. This allows first responders to have more information and coordination before arriving at the scene, as well as for authorities to activate crisis centers and issue alerts. Figure 4 demonstrates a possible scenario for our crisis management application.

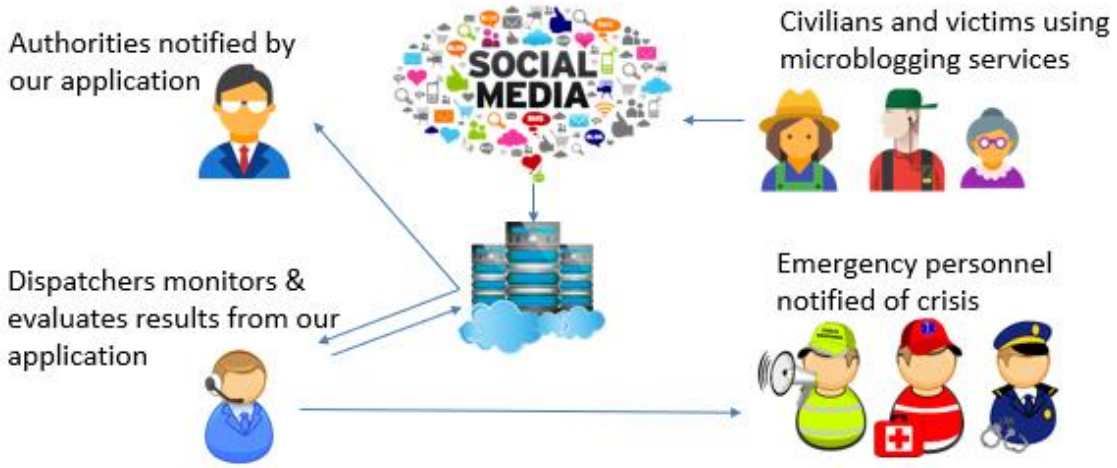

Figure 8: possible use of our system.

4.2 Our Micro-Clustering Process

The first process demonstrated in Figure 7 is our Micro-clustering process. This process connects to a stream or a CSV file and begins incrementally clustering of the incoming data. Internally, this process calls the R wrapper class for the TextClust algorithm, which is responsible for continuously clustering, fading, removing, and sending micro-clusters. Additionally, this wrapper class contains the parameter configuration for the TextClust algorithm. We initially set the parameters values as $r = 0.3593$, $\lambda = 0.7682$, $tgap = 392$, and noticed that our results remained relatively stable for our test dataset. This parameter configuration was previously used by the researchers of the TextClust algorithm. However, with further investigation, we found a more optimal parameter configuration for our dataset: $r = 0.1023$, $\lambda = 0.2102$, $tgap = 252$.

It is worth mentioning that the results of micro-clustering would be quite different depending on which parameter configuration is used. To counterbalance this, our system allows users to change these parameter values. However, further research is needed to determine the optimal parameter configuration for a live data stream; till this research happens we cannot suggest any optimal parameter settings to end-users.

To perform incremental clustering, the micro-clustering process must export its results after processing a certain number of text messages. The cut-off point for the number of text messages is important, because if it is set to too large or too small, it can drastically affect the overall clustering quality as well as the performance of the system. For this reason, we chose the cut-off point as 1000 text messages. The reason is that, when a crisis is underway, hundreds if not tens of thousands of messages are produced in a short period of time. Thus, 1000 messages are not too large to miss important topics, but are not too small to waste a lot of time to perform re-clustering.

Once the cut-off point is reached, a thread is used to transfer the micro-clusters to our Kafka server. Kafka [1], a distributed streaming platform developed to build a real-time data pipeline, was used in conjunction with Zookeeper [2], an open-source server, to pipe micro-clusters from the micro-clustering process to the macro-clustering process, and finally to the database. This process is clearly shown in Figure 7, with Zookeeper and Kafka visible in the center. Using a pipeline, we ensure that all the data processing threads, visualization threads, and database threads are unconcerned with the source of their input data. Moreover, this approach ensures that micro-clustering is performed continuously, and the micro-clusters are exported periodically. In essence, the micro-clustering process is working as a producer, which periodically produces micro-clusters; and the macro-clustering process is working as a consumer, which reads in micro-clusters, re-clusters them, and saves the new macro-clusters in the database. This simple, yet efficient architecture allows for continuous, uninterrupted clustering of live stream data and the ability to visualize and save the clustering results.

4.2.1 TextClust Modifications

To be able to use the TextClust algorithm for our research, we modified certain features of the TextClust algorithm, and we modified the structure of a Micro-cluster. The main modifications are listed below:

- We completely re-modified the entire initial text preprocessing procedure to what was discussed in section 2.2.
- We changed the algorithm‘s tokenization function to one that now works with microblogging data. The tokenizer we used was called tokenize_tweets.

- We modified the internal structure of a micro-cluster to now consist of a list of terms, the occurrence count of each term, the overall micro-clusters weight, the last time the micro-cluster was updated, a list of IDs, a list of geolocation's, and a list of timestamp values.
- And we modified the overall TextClust algorithm to work with our new micro-cluster.

4.3 Our Macro-Clustering Process

Next is our macro-clustering process, and it is composed of two main threads — the *analyzer thread* and the *macro-clustering thread*, seen if Figure 7. Both threads can run concurrently, and all data transfer happens through a shared queue. The *analyzer thread* is responsible for endlessly querying the Kafka server for any newly arrived micro-clusters. Once it receives a new micro-cluster, it places it into the shared queue. On the other hand, the macro-clustering thread can only access micro-clusters through the shared queue.

To ensure safe access to this shared queue, we used a semaphore to guarantee the mutual exclusion between these two threads. However, this precaution is arguably redundant as the *analyzer thread* can only insert data into the queue and the *macro-clustering thread* can only read from the queue. As a result, both threads can run independently of each other.

In the *macro-clustering thread*, we measure the similarity between micro-clusters by using the cosine similarity, which is widely used in text clustering. Review equation (3) in section 2.4 for more details on the cosine similarity.

Additionally, the *macro-clustering thread* was carefully designed to provide end-users with the ability to analyze the text stream data by using different clustering algorithms. It is well-known that there is no single perfect clustering algorithm for all

application domains, which is doubly true for live text stream data. For example, one clustering algorithm could perform well for a short period of time, then suddenly start giving erroneous results. Relatively simple changes in the composition of the data like central tendency, the distribution of text, and semantic context can easily make different clustering algorithms produce quite different results. Subsequently, we developed our system such that users can choose the clustering algorithm in an interactive manner. To achieve this, we integrated two powerful software design patterns — the strategy and factory design patterns [11].

4.3.1 Our Software Design Patterns

The factory design pattern is a creational pattern that provides a mechanism for creating objects without specifying the exact class of the object to be created. In our case, we utilize the factory design pattern to create and instantiate a clustering algorithm. This helps to encapsulate the logic concerned with the instance creation and helps us select an appropriate strategy pattern.

The strategy design pattern, which is a behavioral design pattern, deals with selecting an appropriate algorithm to execute at runtime. This design pattern receives end-user instructions on which algorithm to run.

Additionally, these two design patterns enforce all implemented algorithms to inherit from one interface. This interface provides specific methods that all algorithms must implement, which ensures we always obtain the results we need from the algorithms; in particular, all algorithms must return the following key criteria: the number of clusters found, the clusters themselves with all the tokens (i.e., terms), the individual terms weight, the terms geolocations, the clusters internal evaluation score, the cluster's top 10 terms, and any generated graphs.

In short, by intertwining these two design patterns, we have abstracted away from what algorithm is running. Our system no longer cares what algorithm is running as long as it inherits from our interface. Thus, crisis management personnel or other users can change the macro-clustering algorithm in mid-flow to a more appropriate one. Our system provides users with the ability to view visual graphs and statistical measurements to help them decide whether or not the macro-clustering algorithm is performing in the desired way.

4.3.2 K-Means Algorithm And Our Implementation

When our system initially starts, the default macro-clustering algorithm is k-means. As already mentioned, end-users can dynamically change to a different algorithm at runtime. We cluster the micro-clustering results in the form of their normalized TF–IDF vectors by using k-means. The steps of the k-means clustering algorithm are as follows:

1. Select k initial cluster centroids, each of which represents a cluster.
2. For each document in the whole dataset, compute the similarity with each cluster centroid and assign the document to the closest (i.e., most similar) centroid.
    i. The similarity measure used was equation (4)
3. Recalculate k centroids based on the documents assigned to them.
4. Repeat steps 2 and 3 until convergence.

Another important thing to consider when using k-means is how to determine the optimal number of clusters k. This k value is extremely important to produce a meaningful result.

To determine the optimal k value, we use the silhouette coefficient, introduced in section 2.5.1 and demonstrated in equation (5), which is a comparative value based on the tightness and separation of the clusters created [27]. We initially start clustering with a low

k value, such as 2, then we calculate the silhouette coefficient. If it is below our threshold value 0.7, we increment the *k* value by 1 and cluster again. This step is repeated until the silhouette coefficient threshold is reached or the maximum *k* value (set to 50) is reached. During this clustering process, we continuously save the optimal k-means model — the optimal silhouette coefficient and the optimal *k* value. The final result or outcome of this clustering process is that we obtain a relatively optimal number of clusters, and subsequently, a strong clustering result.

Figure 9 shows the silhouette coefficient calculated for each *k* value. For this scenario the highest silhouette coefficient score was extremely close to 0.7 and the optimal number of clusters was 22. This scenario also demonstrates why we save the optimal k-means model, instead of just using the last result; since the last result was significantly lower than the optimal. Our previous crisis management tool did not take this into account and used only the ending model, not the optimal one.

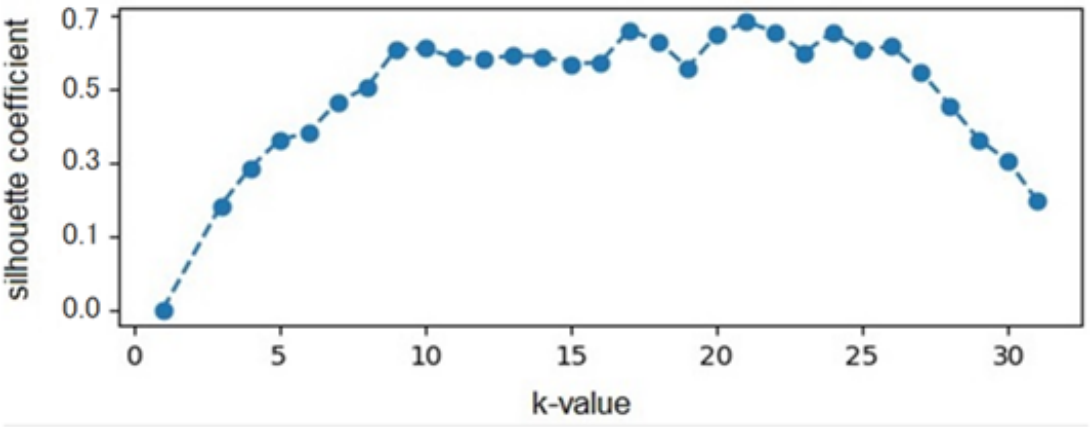

Figure 9: silhouette coefficient vs. k-values.

4.3.3 Agglomerative Hierarchical Clustering

The other macro-clustering algorithm we used was the agglomerative hierarchical clustering. The algorithm typically starts off by letting each object form its own cluster and iteratively merge two most similar clusters, until all the objects are in single cluster or

certain termination conditions are satisfied. The similarity measure we used was cosine similarity; see equation (3). The implementation of this algorithm that we used from the Python library used Ward's minimum variance method [26] as a merging criterion.

Like the case of k-means, we need to determine a reasonable number of clusters. The most reliable way to do it is manually inspecting the hierarchical clustering dendrogram, which is a visualization in the form of a tree showing the order and distances of merges during the hierarchical clustering [13]. If there is a huge jump in merge distance in the dendrogram, we can pick up the corresponding number of clusters. However, this is impractical in a real-time system. Therefore, we used the *elbow* method to automate this procedure.

The elbow method tries to find the clustering step where the acceleration of merge distance is the biggest [13]. For a given sequence of merge distances $< d_1,\ d_2,\ d_3,\ \dots\ , d_n >$ from a dendrogram, the accelerations can be represented as follows:

$$\{\ d_3 - 2d_2 + d_1,\ \dots\ , d_n - 2d_{n-1} + d_{n-2}\ \}$$

From this, we can estimate the number of clusters based on the following formula:

$$\hat{k} = n + 2 - \ argmax_{i \in [3,n]}\ \{\ d_i - 2d_{i-1} + d_{i-2}\}$$

4.4 Results And Discussion

Choosing the right method to evaluate the clustering quality of a clustering algorithm is as important as choosing the right clustering algorithm itself. However, in a stream data environment, it is impractical to use an extrinsic evaluation method because it is nearly impossible to provide the ground truth. Thus, using an intrinsic evaluation method is the only realistic and efficient way to evaluate the clustering quality [20].

The intrinsic measure we used for the k-means clustering algorithm was the

silhouette coefficient [27], and for the agglomerative hierarchical clustering algorithm, we used the Cophenetic correlation coefficient [25]. Readers interested in viewing the mathematical equations used for both of these internal evaluation measures are directed to equation (5) and equation (6), respectively.

At the time of writing, these two clustering algorithms are the only ones we have tested in our system, and they were sourced from the Python scikit-learn library.

4.4.1 K-Means Evaluation

To evaluate the results of the k-means component, of our system, we performed micro-clustering and macro-clustering for 120,000 messages. Figure 10 shows the number of macro-clusters and the silhouette coefficient for each round of macro-clustering.

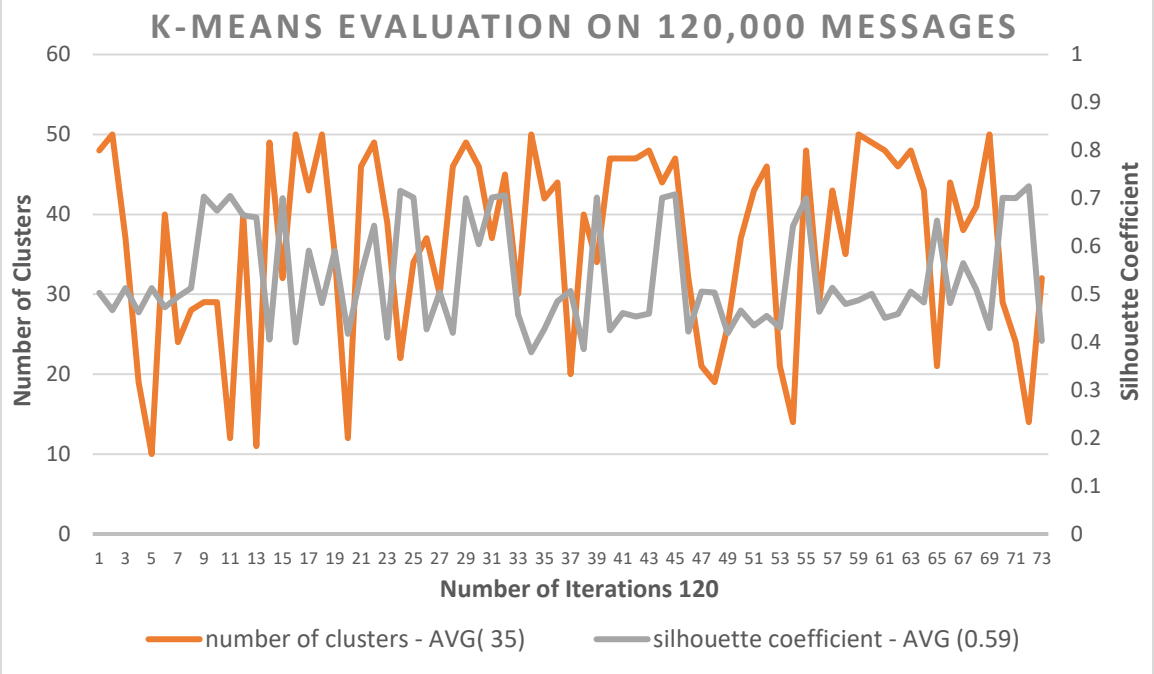

Figure 10: evaluation of k-means clustering results.

The results of our analysis showed that, on average, the silhouette coefficient was above 0.59, and quite often it was even above 0.7, which indicates strong and consistent clustering results. Moreover, only once does the silhouette coefficient drop as low as 0.3. We also

noticed that the cut-off point used in micro-clustering (i.e., the maximum number of messages that are micro-clustered together) influenced the final clustering result. Subsequently, we tested with different values and found that the cut-off point in the rage of [1000, 4000] is reasonable for our dataset, in terms of the quality of the final clustering results and total execution time. Not only do these values produce meaningful results but our system performs efficiently. Lastly, this entire macro-clustering process took only 4.17 seconds on average to re-cluster the micro-clustering results.

4.4.2 Agglomerative Hierarchical Clustering Evaluation

Instead of using the silhouette coefficient (see equation 5), we can evaluate the result of agglomerative hierarchical clustering by comparing the distance between every pair of micro-clusters, contained in the final clustering result, with the distance between the clusters they are assigned to. The *Cophenetic correlation coefficient* (CPCC) that was used for this comparison is a variant of the Pearson correlation coefficient (aka Pearson product-moment correlation coefficient) [25]. In our case, CPCC can be defined as a measure of how accurately the resulting dendrogram preserves the pairwise distance between the objects (i.e., micro-clusters). We used CPCC as a quality measure of macro-clustering results and to help fine-tune our macro-clustering process. If the CPCC score is close to 1, we can consider the clustering result as unquestionably accurate. To see the mathematical equation of CPCC that was used in this evaluation process see equation (6).

To measure the quality and performance of this hierarchical clustering component of our system, we performed micro-clustering for 74,000 messages to generate 74 groups of micro-clusters, and subsequently performed macro-clustering for each group of micro-clusters. Figure 11 shows the number of macro-clusters and CPCC for each round of macro-clustering.

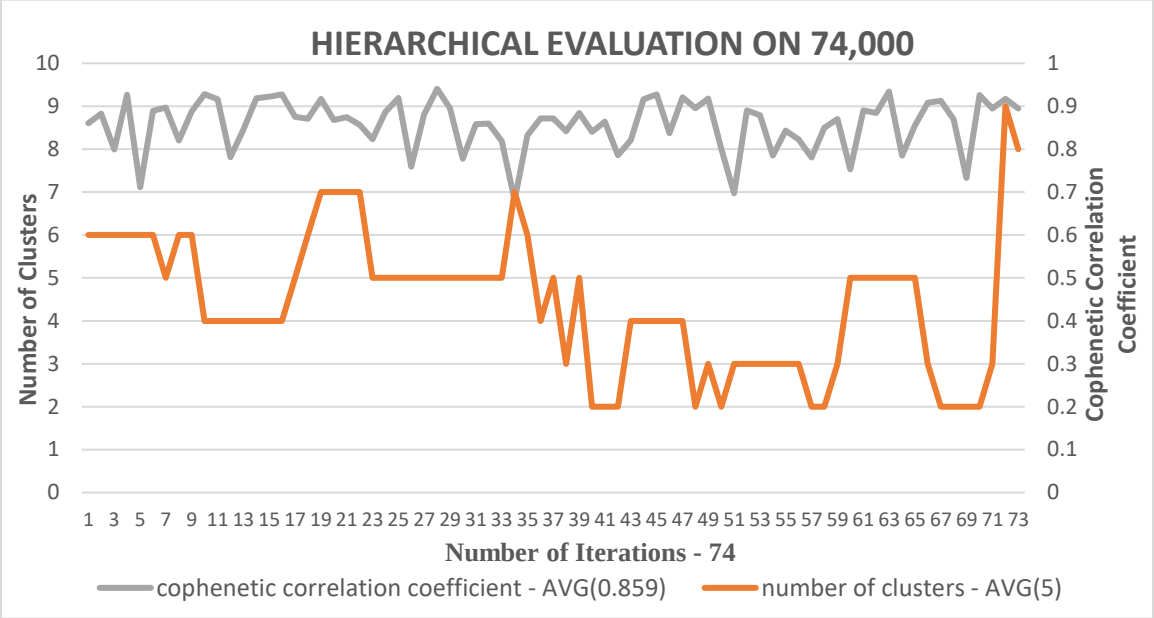

Figure 11: evaluation of hierarchical clustering results

On average, CPCC was 0.859, which indicates that the objects (i.e., micro-clusters) clustered together are undoubtedly close to each other and likely belong to the correct cluster. Moreover, we manually inspected all 74 dendrograms to see if the estimated number of clusters was correct. We found that the elbow method correctly estimated the number of clusters for 93% of our test cases. In the cases where the number of clusters was incorrectly estimated, the error was in the range of plus-minus 2. Thus, we have confirmed the elbow method is unquestionably effective at determining the number of clusters for our data set. Additionally, the dendrograms can be viewed and verified for accurate results, as our system temporarily saves the dendrograms in the database.

4.4.3 Micro-Clustering Evaluation

For the micro-clustering component in our system, we evaluated the average number of messages it can process per second. This evaluation was conducted on an Intel® Core™ i7-4700MQ 2.40GHz with 8.00GB of DDR4 RAM running Windows 10. On average, it could process 20 messages per second; however, we experienced spikes and dips in its

performance which can be explained by the varying length or complexity of individual messages. Based on these positive results, we expect that our system can perform in real-time.

4.5 Our Database Process

For the incremental clustering, how to store the history of previous message clusters is extremely important, as it is required to allow first responders to play back the changes over a period of time. Those stored results can be used for further analysis to see if the crisis could have been detected earlier and how exactly people were using social media to communicate with each other and with emergency personnel during a crisis.

We use MongoDB to store the following information in our database: the results of micro-clustering and macro-clustering, summary statistics, and all the graphs generated. This allows a crisis management personnel to later reanalyze the data by performing consensus clustering and review the snapshots of what happened in order to gain a better understanding of the past crisis. However, since we are working with an infinite stream of data, eventually our database would be bogged down with outdated and useless results. Therefore, we provide a user-specified threshold to remove old and unwanted data.

4.6 Dataset Used

In order to fairly compare our new system against the old one [23], we decided to evaluate our approach using the same dataset from the 2011 IEEE VAST Challenge, which is related to the 2011 IEEE Conference on Visual Analytics, Science, and Technology (IEEE VAST). The mini-challenge 1 dataset (Geospatial and Micro-blogging Characterization of an Epidemic Spread) [24] is similar to that of a real-life crisis and in the same format as a microblog captured from Twitter. The sample dataset, in a comma-separated values (CSV) file, contains user ID field, timestamp field, geolocation field, and text message field.

Originally, the sample IEEE VAST challenge dataset had geolocation's that corresponded to a hypothetical world; however, it was deemed more relevent to set geolocations to a real world location. Subsequently, all data points displayed are now located within a 60,000-mile radius of New York City. Nevertheless, if a different CSV file or data stream is to be used, our system would display different geolocation tags accordingly, without any necessary reconfiguration.

Our current system is designed to work with either a CSV file or a live data stream. In order to read a live data stream, our system utilizes an R package called *stream* [12] and *rpy2* [9] to convert the R package into Python accessible code. Additionally, our system works seamlessly even if not all the potentially required data fields are present. That means, if the data stream or CSV file does not contain all of the data fields, our system will automatically insert default values for those missing fields. For example, if a user provided only a user ID and text message for the stream or CSV file, our system automatically fills in default geolocation coordinates.

One could argue that allowing our system to auto fill in missing geolocations could skew our results. However, this is not the case as our system only performs analysis on the messages themselves. Therefore, the results would still show the main topics taking place in the live data stream, and crisis management personnel can decide to exclude clustering results that do not have geolocation’s in their area. This feature allows our system to be used even if the microblogging service does not support geo-tagging or geolocation.

In summary, the only field that is required in the data stream or CSV file is the text message field. However, to truly take advantage of our system, we strongly suggest that the data be comprised of at least user ID, geolocation, and text message fields.

CHAPTER 5
VISUALIZATION OF DATA

5 Visualization

The goal for this project was to create a tool that would allow first responders and crisis management personnel to seamlessly and easily respond to a developing crisis by tapping into social media and microblogging services. Unquestionably, this tool needs to have a useful visualization component. This component must provide a streamlined, easy to use, and useful way to provide at-a-glance information to those users, so that they can tell if a crisis is underway and help them determine how best to respond. Furthermore, this tool must be cross-platform independent.

To achieve this goal, we developed an interactive web-based application. Using a web application for the visualization of macro-clusters provides end-users with the portability and flexibility they need to access our system on any device.

Our web application works by periodically receiving macro-clusters and displaying the top 10 topics in each macro-cluster. All the top 10 topics of a macro-cluster are displayed as a word cloud, which is an image composed of words appearing in the cluster, and the size of each word in the image indicates its frequency. The word clouds are displayed in different colors to help distinguish them.

Each word cloud has the geolocations associated with its main topic whose weight is the highest. When an end-user clicks on a word cloud, all the geolocations corresponding to the top term in that cluster are displayed as pins on the map. The map can be dragged to different areas by users, and the map can be viewed in different orientations (e.g., zoomed in and out). Figure 8 clearly demonstrates this by showing the term '#brushfires' is the

main topic in that cluster, and also shows the geolocations of the messages being displayed on the map. Lastly, when a new macro-clustering result becomes available, the display is refreshed with the new set of macro-clusters.

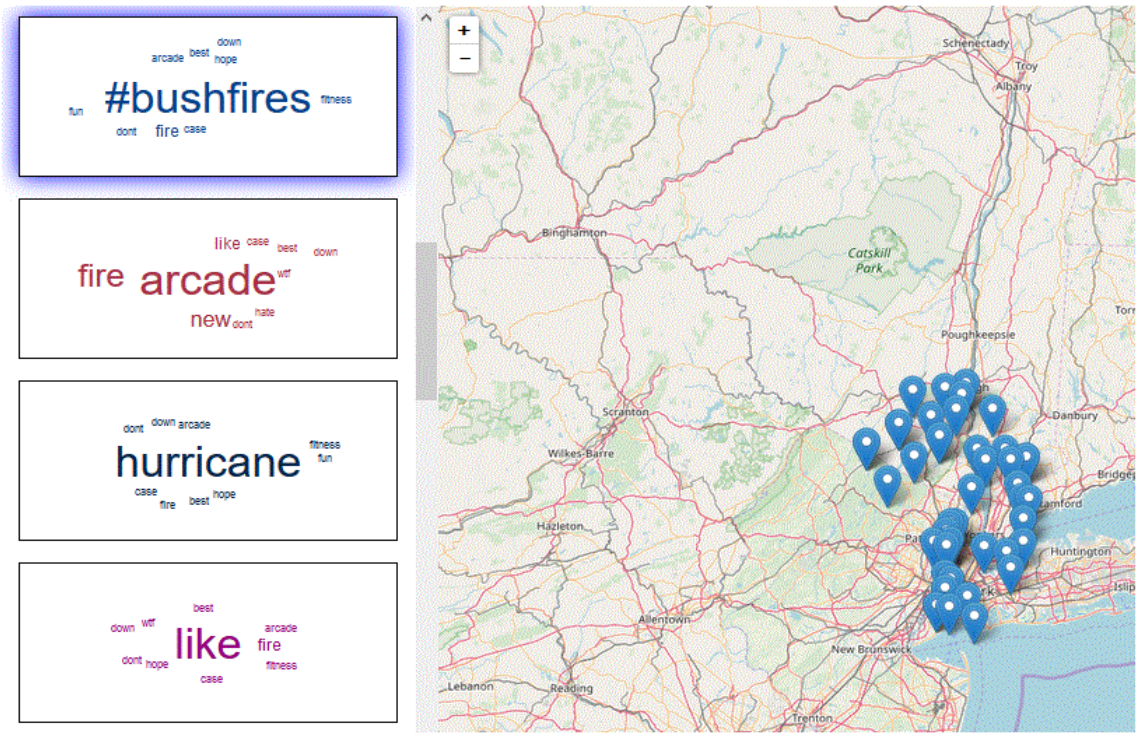

Figure 12: our web application visual interface.

This visualization component was built by using the following open-source technologies:

- JQCloud [17] was used to build word clouds.
- Leaflet.js [14] was used to create an interactive map.
- Express [10], which is a minimalist web framework, was used together with Node.js [22] to provide a web interface for the visualization of the data.
- Jade [15], a template language for html, in conjunction with JavaScript, provides the internal code used for the web-based application.
- JSON format is used for all message passing between processes and threads.

CHAPTER 6
CONCLUSIONS AND FUTURE RESEARCH TOPICS

6.1 Conclusion

In this thesis, we introduced our new text stream clustering tool for crisis management personnel. This tool could be easily used by first responders and crisis management personnel to quickly determine if a crisis is happening, where it is concentrated, and what resources are best to deploy to the situation. Our tool uses the well-known two-phase clustering approach, composed of micro-clustering and macro-clustering, and allows end-users to switch between different macro-clustering algorithms at runtime in order to improve the overall clustering result.

For accurate clustering, we used intrinsic evaluation methods to determine the number of clusters, optimize the clustering result, and provide users with immediate feedback. Our system can process a large number of text messages efficiently by using multithreading. Additionally, our tool is designed to be scaled up to a distributed system, running on multiple servers in parallel, and easily integrated with other related systems. Our system also stores the incremental clustering results as well as useful statistical information in a database. This allows crisis management personnel to view historical data and potentially see if a crisis could have been detected earlier and how exactly people were using social media and microblogging services to communicate about the ongoing crisis. Lastly, for the visualization component of our system we used a word cloud to graphically display each cluster and their top 10 most frequent words.

In short, we posit that our system would be a useful tool for various crisis management applications for the following reasons:

- Our system allows at-a-glance information to be displayed throughout the evolution of a crisis.
- Our system can process an infinite stream of text messages efficiently and compactly.
- Integrating our system into a crisis management application would be extremely cost-effective as all components are open source, all components are streamlined, all components can run independently, and all components conserve memory and secondary storage space.
- Integrating our system into a crisis management application would be trivially easy as the architecture of our system is designed to be easily portable, easily enhance-able, and easily upgradable.
- Our system can change macro-clustering algorithms dynamically to improve the overall clustering accuracy.
- And lastly, our system is extremely customizable.

6.2. Research Contributions

My specific contributions are as follows:

- Researched and analyzed stream clustering algorithms.
    - Defined all criteria that the stream clustering algorithm must exhibit.
- Configured, modified, and redesigned the TextClust algorithm.
    - Learned R and Rcpp package, which provides C++ classes that greatly facilitate interfacing C/C++ code in R packages.
- Researched strategies for seamlessly integrating the TextClust algorithm into our Python environment.

- Learned rpy2 library, which provides a Python interface to the R language.

- Planning, designing, and developing the entire system architecture.
  - Set up a private repository on GitHub.
  - Learned Kafka, which is our Web and streaming framework.
  - Created a comprehensive developer's documentation with the use of Doxygen for our system.
  - Designed and implemented our database and created queries for retrieving data.
  - Planned and implemented strategies for testing every component of our system.
- Performed the evaluation of the tool using our datasets, and then refined our system's design and implementation.

6.3 Further Research Topics

6.3.1 Visualization Extension

This visualization component is just the initial way for crisis personnel and first responders to view relevant data. We are currently working on extending this visualization component by incorporating many of the features that are already supported by our underlying system architecture. For example, the controller has mechanisms for querying the database, changing parameters, and updating user preferences. However, our current visualization component in the view does not take full advantage of this yet.

Another crucial enhancement we are planning to develop specifically for the web-based component is to implement a hierarchical grouping strategy for the pins. Currently, all pins belonging to a specific cluster are displayed on our interactive map, and if a specific cluster contains hundreds, if not thousands, of messages, then the number of pins displayed

could be overwhelming for crisis personnel. Therefore, we propose developing a mechanism that would allow us to reduce the number of pins displayed based on the current map's zoom level. This means that when a user zooms out of the map, less pins are displayed for a specific region and more pins are displayed when a user zooms into a specific area. Ultimately, if the user zooms out to a global perspective or country perspective we would only display one pin for that specific cluster. This technique will allow us to display multiple clusters on our interactive map without making the map too cluttered.

Lastly, we are developing a 3-D view component to visualize the data in a completely unique and interactive way.

6.3.2 Anomaly Detection In Microblogging

As of 2019, Twitter has approximately 330 million active users, all contributing to the rapid production of data. This data is unstructured and contains differing topics (such as news stories, politics, sports, fashion, etc.). The data produced by twitter users is available in the form of data streams. These data streams allow researchers to develop analytical tools for gleaning useful information from microblogging services. However, the increasing popularity and heavy use of microblogging services, like Twitter, has made it hard for researchers to collect useful information in a timely manner. Moreover, a significant portion of this data is irrelevant to the main interests of researchers.

Currently, our system is designed to read and cluster a live data stream in order to predict potential disasters and crises. However, in a real-time, live data stream, most of the data being processed by our system would be irrelevant to the specific topic of crisis detection that we are interested in. Therefore, we are looking into ways to ensure that our system would not waste resources by clustering and analyzing millions of messages that

are not important to our topic. The technique we are currently investigating is stream anomaly detection [29, 30, 31, 32]. The current enhancement we are planning to implement is to utilize local event reports, local news outlets, and global media reactions to detect spatial anomalies and then adjust our analysis appropriately. Additionally, our goal is to continuously analyze the spatial and temporal attributes of the data stream and detect contextual and collective anomalies. Detected anomalies could be either filtered out in our system, in order to save storage space and processing time, or solely be used in our analysis depending on which type of anomaly is detected.

6.3.3. Interaction With Emergency Responders

Once we have developed these new visualization features and system enhancements, we plan on directly collaborating with emergency responders. The goal of this collaboration will be to receive critical feedback from the end-users of our system. Based off the outcome of this collaboration, we will fine-tune our system to meet their needs. During this collaboration, we plan on utilizing a real-life crisis data set as this will ensure that emergency personnel will see the benefit or potential flaws of our system during this evaluation process. This is an important future step in our research, as the end goal is to provide first responders and emergency personnel with a tool that helps them to respond faster to a developing crisis, helps them save more lives, and helps them reduce needless suffering.

REFERENCES

[1] Apache Kafka, kafka.apache.org/

[2] Apache Zookeeper, zookeeper.apache.org/

[3] M. Bunz, “In Haiti Earthquake Coverage, Social Media Gives Victim a Voice,” 2010. www.guardian.co.uk/media/pda/2010/jan/14/socialnetworking-haiti

[4] M. Carnein, D. Assenmacher, and H. Trautmann, “Stream Clustering of Chat Messages with Applications to Twitch,” Advances in Conceptual Modeling (Proc. of ER 2017 Workshops), LNCS 10651, 2017, pp. 79−88.

[5] C. C. Aggarwal and P. S. Yu, On Clustering Massive Text and Categorical Data Streams, Knowledge and Information Systems, vol. 24, no. 2, 2010, pp. 171−196.

[6] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu, “A Framework for Clustering Evolving Data Streams,” Proc. of the 29th Int’l Conf. on Very Large Data Bases, 2003, pp. 81−92.

[7] C.C. Aggarwal, A. Hinneburg, D.A. Keim, “On the Surprising Behavior of Distance Metrics in High Dimensional Space,” In: J. Van den Bussche and V. Vianu (eds), ICDT 2001, LNCS vol 1973, Springer, 2001, pp. 420−434.

[8] C. C. Aggarwal, 2014. “Mining Text and Social Streams: A Review,” ACM SIGKDD Explorations Newsletter, vol. 15, no. 2, 2014, pp. 9−19.

[9] Documentation for rpy2, Retrieved from rpy2.readthedocs.io/en/version_2.8.x/

[10] Express, expressjs.com.

[11] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Professional, 1994.

[12] M. Hahsler, M. Bolanos, J. Forrest, M. Carnein, and D. Assenmacher, “stream:

Infrastructure for Data Stream Mining," 2015, cran.r-project.org/web/packages/stream/

[13] J. Hees, "SciPy Hierarchical Clustering and Dendrogram Tutorial," 2015, joernhees.de/blog/2015/08/26/scipy-hierarchical-clustering-and-dendrogram-tutorial/

[14] Leaflet: An Open-source JavaScript Library for Mobile-friendly Interactive Maps, leafletjs.com/

[15] Jade, www.jadeworld.com/

[16] J. A. Silva, E. R. Faria, R. C. Barros, Eduardo R. Hruschka, A. C. de Carvalho, and J. Gama, "Data Stream Clustering: An Aurvey," ACM Computing Survey, vol. 46, no. 1, 2013, pp. 9−19.

[17] JQCloud, github.com/lucaong/jQCloud

[18] K. A. Lachlan, P. R. Spence, X. Lin, K. Najarian, and M. D. Greco, "Social Media and Crisis Management: CERC, Search Strategies, and Twitter Content," Computers in Human Behavior, vol. 54, 2016, pp. 647−652.

[19] R. C. Martin, "Design Principles Page," Retrieved from condor.depaul.edu/dmumaugh/OOT/Design-Principles/

[20] M. Hassani and T. Seidl, "Using Internal Evaluation Measures to Validate the Quality of Diverse Stream Clustering Algorithms," Vietnam J. of Computer Science, vol. 4, no. 3, 2017, pp. 171−183.

[21] C. C. Aggarwal, Data Mining: The Textbook, Springer, 2015.

[22] Node.js, nodejs.org/en/

[23] S. A. Barnard, S. M. Chung, V. A. Schmidt, "Content-based Clustering and

Visualization of Social Media Text Messages," Proc. of Int'l Conf. on Data and Software Engineering (ICoDSE 2017), 2017.

[24] 2011 IEEE Conference on Visual Analytics Science and Technology (VAST) Mini Challenge 1 Dataset: Geospatial and Microblogging Characterization of an Epidemic Spread. Available at hcil2.cs.umd.edu/newvarepository/benchmarks.php#VAST2011

[25] S. Kumar and D. Toshniwal, (2016), "Analysis of Hourly Road Accident Counts Using Hierarchical Clustering and Cophenetic Correlation Coefficient (CPCC)," Journal of Big Data, vol. 13, article# 13, 2016.

[26] J. H. Ward, "Hierarchical Grouping to Optimize an Objective Function," Journal of the American Statistical Association, 58, 1963, pp. 236-244.

[27] P. J. Rousseeuw, "Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis," Journal of Computational and Applied Mathematics, vol. 20, 1987, pp. 53−65.

[28] Spärck Jones, Karen. "A Statistical Interpretation of Term Specificity and Its Application in Retrieval." *Journal of Documentation* vol. 28, no. 1 (1972): 16.

[29] H. Bosch, D. Thom, F. Heimerl, E. Püttmann, S. Koch, R. Krüger, M. Wörner, and T. Ertl, "Scatterblogs2: Real-time Monitoring of Microblog Messages through User-guided Filtering," IEEE Trans. on Visualization and Computer Graphics, vol. 19, no. 12, 2013, pp. 2022−2031.

[30] J. Huang, M. Peng, H. Wang, J. Cao, W. Gao, and X. Zhang, "A Probabilistic Method for Emerging Topic Tracking in Microblog Stream," World Wide Web, vol. 20, no. 2, 2017, pp. 325−350.

[31] W. Yang, G. Shen, W. Wang, L. Gong, M. Yu, and G. Dong, “Anomaly Detection in Microblogging via Co-clustering,” J. of Computer Science and Technology, vol. 30, no. 5, 2015, 1097−1108.

[32] J. Guzman and B. Poblete, “On-line Relevant Anomaly Detection in the Twitter Stream: An Efficient Bursty Keyword Detection Model,” Proc. of the ACM SIGKDD workshop on Outlier Detection and Description 2013, pp. 31−39.